Advanced Virtual Memory for Exascale*

Kamil Iskra, Kazutomo Yoshii, Rinku Gupta, Pete Beckman

Mathematics and Computer Science Division
Argonne National Laboratory
9700 South Cass Avenue, Argonne, IL 60439, USA
{iskra,kazutomo,rgupta,beckman}@mcs.anl.gov

1 Background

By nearly every metric—cost, power consumption, capacity, bandwidth, and latency—memory is emerging as one of the most constrained resources on compute nodes of HPC systems [12]. These constraints will force future systems to abandon flat SMP memory space in favor of a more distributed NUMA design, even within a socket. With exponentially increasing numbers of cores per node, the overhead of global cache coherence is likely to exceed the usefulness of this feature, at which point cache coherence will be limited to multiple separate coherence domains within a CPU. Experimental designs such as Intel SCC [13] or NVIDIA Echelon [6] are early examples of this trend.

Power is also a consideration for translation lookaside buffer (TLB) implementations; for example, in Intel StrongARM, TLB consumes 17% of the chip power [10]. For that reason, on many-core heterogeneous systems, lightweight "throughput" cores are likely to lack advanced capabilities such as hardware page walking support for TLB misses. We have seen the consequences on Blue Gene systems when running a kernel that uses paged memory, where irregular access patterns can result in a performance decrease by an integer factor compared with TLB-miss-free execution [16].

HPC programming models are becoming more diverse. Partitioned Global Address Space (PGAS) languages such as UPC [19] and Co-Array Fortran [5] explicitly control data locality and distribution; their memory model places requirements on memory management different from those of the MPI-only model. Dynamic languages such as Python are also gaining traction as a mechanism to assemble an application from many pre-existing components. Emerging capabilities for thread level speculation [8, 17, 18] or transactional memory [9, 15] place even more requirements, as do programmable prefetch engines, complex alignment requirements, and the possibility of stacked memory on the CPU. The integration of NVRAM into the node is yet another consideration.

Clearly, a considerably more advanced memory management system is needed.

2 Advanced Virtual Memory

We expect the exascale systems to keep the basic concept of virtual memory; it is simply too convenient an abstraction from the point of view of programmability and resource management to be abandoned readily. It is critical to fault isolation and protection between core system management functions and application components, as well as to managed access to hardware that could be integrated as an extension to the memory system, such as NVRAM and networking queues.

Paged memory, on the other hand, will not be generally sustainable in HPC contexts because of the TLB miss overheads, to which HPC applications can be very sensitive. Increasing the TLB size makes the power problem worse, and continuing to increase the page size results in greater memory fragmentation. Still, paged memory enables a number of useful optimizations such as memory-mapped I/O, zero pages, or copy-on-write, and a number of useful scripting or debugging tools depend on it. Thus, it cannot be abandoned completely.

We believe that the best solution for future systems is a hybrid scheme featuring multiple memory management models within different regions of the virtual address space. Using different page sizes and TLB purge policies can provide both maximum, TLB-miss-free performance for more static memory regions, such as program text, heap, or stack, and the flexibility of paged memory needed by dynamically loaded shared libraries and lightweight threading. We call this scheme "advanced virtual memory"; in addition to extensive software effort required on the OS kernel side for a seamless integration, the hardware (specifically the memory management unit and the TLB) needs to be flexible enough to support multiple page sizes and a combination of static and dynamic TLB entries. This infrastructure will need an API to determine, in coordination with the runtime system and applications,

*This work was supported by the Office of Advanced Scientific Computer Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357.

which regions of the virtual address space are thread-local (private) or read-only and which are actually meant to be shared between threads in read-write mode. On current SMP systems, this is a largely artificial distinction; but in the future, this information could be used to optimize the cache coherence protocol on hardware that supports it, so that the overhead of cache snooping is limited to the CPU cores and memory pages that need it.

It should also be possible, for some HPC data structures, to turn off the cache coherence protocol completely or to set the "tile size" for cores sharing a snooping cache. Consider application data structures that are shared but infrequently modified. Core-local caches can still provide fast access to the data, but software-based invalidation would be required to handle updates to the structure. In fact, the first Blue Gene (BG/L) used this scheme because cache coherence hardware was not included in the chip. Software cache coherence was the only option for SMP-like operation of user-level threads [2]. Another approach would be to duplicate frequently accessed read-only data using local, noncoherent memory, if such is available. The OS could perform such merges and splits transparently, depending on memory load and access pattern. CPU cache could be explicitly partitioned between "local" and "main" memory in order to prevent local memory accesses from purging the lines holding main memory data; this approach is already possible on PowerPC CPUs, but no APIs currently exists to take advantage of this feature. We need to explore how to provide such advanced functionality to applications with minimal modification to the application code.

For large, many-core CPUs an explicit intranode message passing system that uses either polling or interprocessor interrupts could provide the functionality needed to build explicit APIs for sharing data structures. Since DMA engines prefer contiguous pinned pages, it is also important that such interfaces reflect whether the region will be the target of a DMA from either another core or the interconnect.

3 Related Work

In mainstream OS kernels, effective transparent *superpage* management system that can utilize larger physical pages to reduce TLB misses has been implemented by Navarro et al. [14] in the FreeBSD kernel. Linux kernel has had support for larger pages through *hugetlbfs* [4] for a long time, but it has been far from convenient to use. Recent work of Arcangeli [3] adds the much-needed transparency; that support, however, is limited to anonymous (dynamically allocated) memory and provides essentially a best-effort infrastructure that does not guarantee performance predictability or repeatability.

In HPC space, predictable, high memory performance through the use of large pages is provided by lightweight kernels such as CNK [1] on Blue Gene and Catamount [11] on Cray; such kernels have a number of other limitations, however, that do not make them the best choice for managing complex exascale nodes. Shmueli et al. [16] evaluated Linux's hugetlbfs on the compute nodes of BG/L using a user-space libhugetlbfs [7] wrapper and found it to provide many of the memory performance benefits of lightweight kernels. At Argonne, we have developed Big Memory [20–22], a memory region covered by very large, semi-static TLB entries. A computational process can be transparently mapped into Big Memory to achieve performance parity with IBM's CNK, while the rest of the node can keep using paged memory. Because Big Memory is physically contiguous, it can be used for DMA-based internode communication on Blue Gene/P.

4 Summary

Challenges addressed: Memory hierarchy, specifically controlling overheads and runtime/application management of memory.

Maturity: As much of the complexity involves combining two well-tested techniques, static TLB and paged memory, we consider the approach to be mature.

Uniqueness: HPC is the driving force behind reducing the overheads of memory access. Related efforts have been undertaken in mainstream OS communities, but the solutions they provide are incomplete and come without performance guarantees, because extending them to cover the full spectrum of possible circumstances that can occur in a complex multiuser environment is hard. A more limited solution targeted to HPC platforms and applications is far more likely to be successful.

Novelty: To the best of our knowledge, no existing solution provides the combination of low overhead, flexibility, and performance guarantees we expect to be necessary by HPC workloads at exascale.

Applicability: To varying extent, every application would benefit from lower memory access overheads; thus, a comprehensive solution would probably be welcome by other communities. As indicated earlier, however, we advocate an HPC-limited solution, which would potentially not cover a full range of multiuser environment use cases (such as the use of swap space), limiting its appeal to other communities.

Effort: The level of effort would vary significantly depending on the choice of the OS kernel and the desired level of integration and applicability outside of core HPC space, but the bulk of the work could conceivably be carried out by two people and an advisor/supervisor within three to five years.

References

[1] N. Adiga et al. An overview of the Blue Gene/L supercomputer. In *Supercomputing '02: Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*, Los Alamitos, CA, 2002. IEEE Computer Society Press.

[2] G. Almási, L. R. Bachega, S. Chatterjee, M. Gupta, D. Lieber, X. Martorell, and J. E. Moreira. Enabling dual-core mode in Blue Gene/L: Challenges and solutions. In *SBAC-PAD*, pages 19–27, 2003.

[3] A. Arcangeli. Transparent hugepage support. https://events.linuxfoundation.org/slides/2011/lfcs/lfcs2011_hpc_arcangeli.pdf.

[4] K. Chen, R. Seth, and H. Nueckel. Improving enterprise database performance on Intel Itanium architecture. In *Proceedings of the Linux Symposium*, pages 98–108, Ottawa, ON, Canada, July 2003.

[5] Co-Array Fortran. http://www.co-array.org/.

[6] NVIDIA Echelon. http://www.nvidia.com/content/PDF/sc_2010/theater/Dally_SC10.pdf.

[7] D. Gibson and A. Litke. libhugetlbfs. http://sourceforge.net/projects/libhugetlbfs.

[8] L. Hammond, M. Willey, and K. Olukotun. Data speculation support for a chip multiprocessor. *SIGOPS Operating Systems Review*, 32(5):58–69, 1998.

[9] E. H. Jensen, G. W. Hagensen, and J. M. Broughton. A new approach to exclusive data access in shared memory multiprocessors. Technical Report UCRL-97663, Lawrence Livermore National Laboratory, Livermore, CA, Nov 1987.

[10] T. Juan, T. Lang, and J. J. Navarro. Reducing TLB power requirements. In *Proceedings of the 1997 International Symposium on Low Power Electronics and Design*, ISLPED '97, pages 196–201. ACM, 1997.

[11] S. M. Kelly and R. Brightwell. Software architecture of the light weight kernel, Catamount. In *Proceedings of the 47th Cray User Group Conference*, Albuquerque, NM, May 2005.

[12] P. Kogge, K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, J. Hiller, S. Karp, S. Keckler, D. Klein, R. Lucas, M. Richards, A. Scarpelli, S. Scott, A. Snavely, T. Sterling, R. S. Williams, and K. Yelick. Exascale computing study: Technology challenges in achieving exascale systems. Technical report, DARPA IPTO, AFRL, Sept. 2008.

[13] T. G. Mattson, M. Riepen, T. Lehnig, P. Brett, W. Haas, P. Kennedy, J. Howard, S. Vangal, N. Borkar, G. Ruhl, and S. Dighe. The 48-core SCC processor: the programmer's view. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10. IEEE Computer Society, 2010.

[14] J. Navarro, S. Iyer, P. Druschel, and A. Cox. Practical, transparent operating system support for superpages. In *Proceedings of the 5th ACM Symposium on Operating System Design and Implementation*, volume 36 of *ACM SIGOPS Operating Systems Review*, pages 89–104, Boston, MA, Dec. 2002.

[15] N. Shavit and D. Touitou. Software transactional memory. In *PODC '95: Proceedings of the Fourteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 204–213. ACM, 1995.

[16] E. Shmueli, G. Almasi, J. Brunheroto, J. Castanos, G. Dozsa, S. Kumar, and D. Lieber. Evaluating the effect of replacing CNK with Linux on the compute-nodes of Blue Gene/L. In *ICS '08: Proceedings of the 22nd Annual International Conference on Supercomputing*, pages 165–174, New York, 2008. ACM.

[17] G. S. Sohi, S. E. Breach, and T. N. Vijaykumar. Multiscalar processors. In *ISCA '95: Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pages 414–425. ACM, 1995.

[18] J. Steffan and T. Mowry. The potential for using thread-level data speculation to facilitate automatic parallelization. In *HPCA '98: Proceedings of the 4th International Symposium on High-Performance Computer Architecture*, page 2, Washington, DC, 1998. IEEE Computer Society.

[19] Unified Parallel C. http://upc.gwu.edu/.

[20] K. Yoshii, K. Iskra, P. C. Broekema, H. Naik, and P. Beckman. Characterizing the performance of Big Memory on Blue Gene Linux. In *2nd International Workshop on Parallel Programming Models and Systems Software for High-End Computing*, pages 65–72, Los Alamitos, CA, Sept. 2009. IEEE Computer Society.

[21] K. Yoshii, K. Iskra, H. Naik, P. Beckmanm, and P. C. Broekema. Performance and scalability evaluation of "Big Memory" on Blue Gene Linux. *The International Journal of High Performance Computing*, 2010.

[22] K. Yoshii, H. Naik, C. Yu, and P. Beckmanm. Extending and benchmarking the "Big Memory" implementation on Blue Gene/P. In *Proceedings of the International Workshop on Runtime and Operating Systems for Supercomputers (ROSS'11)*, pages 65–72, May 2011.